Desain dan Analisis Algoritma Dijkstra dan Metode *Visibility Graph Naive* untuk Penyelesaian Persoalan *Spoj The Archipelago*

Reva Yoga Pradana, Victor Hariadi, Rully Soelaiman
Jurusan Teknik Informatika, Fakultas Masing-masing, Institut Teknologi Sepuluh Nopember (ITS)
Jl. Arief Rahman Hakim, Surabaya 60111 Indonesia
e-mail: rully@if.its.ac.id

***Abstrak*—Permasalahan pada SPOJ 780 *The Archipelago* adalah sebagai berikut. Diberikan sekumpulan pulau yang memiliki beberapa terminal dan area terlarang yang tidak boleh dilewati selama jalur darat di pulau tersebut. Kemudian, dicari kemungkinan rute terpendek yang menghubungkan terminal di suatu pulau ke terminal di pulau yang lain, tanpa melewati area terlarang manapun, dan total jarak pada 1 pulau harus dibulatkan ke atas. Untuk menyelesaikan permasalahan tersebut, dibutuhkan algoritma Visibility Graph Navie, untuk mengetahui titik-titik mana saja yang dapat dilalui, dan algoritma Dijkstra untuk mengetahui rute terpendeknya. Selain itu, juga dilakukan beberapa optimasi untuk mempercepat kinerja sistem, yaitu dengan cara penyederhanaan penggambaran entitas halangan, pengurangan jumlah *vertex* yang harus dicek visibilitasnya, dan penggunaan *backtracking* dari *vertex* tujuan hingga *vertex* sumber untuk mengetahui total jarak pada tiap pulau.**

Cara kerja dari sistem adalah sebagai berikut. Diberikan masukan sesuai deskripsi permasalahan. Setelah masukan berhasil diolah, sistem akan memanggil algoritma Dijkstra. Pada tiap *current vertex* yang dihasilkan di dalam Dijkstra, sistem akan memanggil algoritma Visibility Graph Naive, untuk mengecek visibilitas dari vertex tersebut dengan vertex lain yang se-pulau, agar dapat diketahui vertex mana saja yang menjadi tetangganya. Dijkstra akan berhenti saat *current vertex* merupakan *vertex* tujuan. Kemudian saat Dijkstra telah berhenti, sistem akan menghitung total jarak pada tiap pulau, total jarak keseluruhan, dan menyimpan posisi yang dipilih pada rute terpendek

Hasil akhir menunjukkan bahwa sistem dapat berjalan cukup cepat berkat optimasi yang baik, dengan waktu rata-rata sebesar 0.2763 detik dan memori rata-rata 10.533 MB.

***Kata Kunci*— Graph with Obstacles, Computational Geometry, Shortest Path, Visibility Graph, Dijkstra**

I. INTRODUCTION

Permasalahan yang diangkat dalam riset ini adalah persoalan dari situs SPOJ dengan nomor soal 780, dan kode soal ARCHPLG, dengan judul "The Archipelago". Dalam permasalahan tersebut, terdapat sebuah negara fiktif bernama *Byteland*. *Byteland* memiliki beberapa pulau berbentuk persegi panjang. Beberapa pulau dapat terhubung oleh jalur kapal ferry.
Setiap pulau digambarkan dalam bidang geometri, yang mana memiliki sistem koordinat tersendiri di tiap pulaunya.

Dalam setiap pulau, terdapat 2 elemen, yaitu: area terlarang yang berbentuk persegi panjang dan terminal. Area terlarang tidak boleh dilewati dalam pengambilan rute terpendek. Terminal sendiri merupakan pelabuhan dari Kapal Ferry.

Setelah ditentukan karakteristik dari kepulauan *Byteland*, maka kemudian terdapat permasalahan yang harus diselesaikan, yaitu total jarak rute terpendek dari suatu terminal ke terminal yang lain, dengan ketentuan-ketentuan yang telah ditetapkan sebelumnya. Selain itu juga dicari posisi mana saja yang dipilih pada saat pemilihan rute terpendek.

Dalam permasalahan tersebut, akan ada masukan berupa T yang merupakan jumlah kasus uji. Kemudian pada tiap kasus uji, akan terdapat deskripsi tiap pulau, tiap jalur kapal ferry, dan permasalahan rute terpendek yang harus diselesaikan, dari suatu terminal ke terminal yang lain. Pada deskripsi tiap pulau, terdapat masukan berupa deskripsi tiap terminal dan tiap area terlarang. Tiap terminal dideskripsikan nama dan posisinya dalam bentuk koordinat x dan y. Sedangkan tiap area terlarang dideskripsikan dalam bentuk persegi panjang, membentang dari koordinat (x_1, y_1) hingga (x_2, y_2). Kemudian, tiap jalur kapal ferry dideskripsikan kedua terminal penghubungnya, pulau letak tiap terminalnya, serta jarak jalur tersebut.

Setelah deskripsi tiap masukan, sistem harus mengeluarkan keluaran sesuai dengan ketentuan yang ada. Pertama, sistem akan mencetak nomor kasus ujinya. Kemudian, sistem akan mencetak total jarak terpendeknya. Setelah itu, dicetak posisi-posisi yang dipilih dalam pemilihan rute terpendek. Untuk ketentuan rute terpendeknya, diitung berdasarkan total jarak pada jalur kapal ferry yang ditempuh, serta hasil pembulatan pada total rute di tiap pulau yang dilewati. Kemudian untuk pencetakan posisinya, apabila posisi yang dipilih berupa terminal, maka dicetak nama terminal dan nama pulaunya. Namun apabila posisi yang dipilih bukan berupa terminal, maka dicetak posisinya, dalam bentuk koordinat x dan y[1].

II. TINJAUAN PUSTAKA

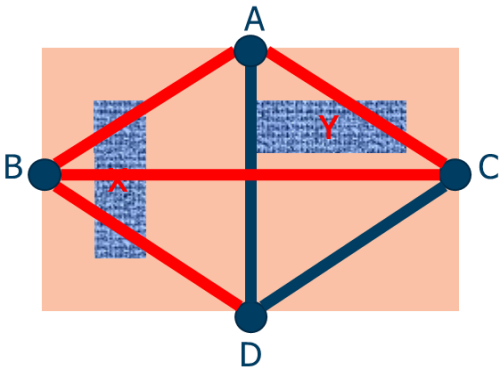
A. Visibility Graph

Visibility Graph digunakan untuk menentukan apakah suatu *vertex visible* / dapat dijangkau oleh *vertex* yang lain. Apabila 2 buah *vertex* saling *visible*, maka dapat dibentuk *edge* antar keduanya. Pada Gambar 1, garis merah melambangkan kedua *vertex* saling tidak *visible*, sedangkan garis biru kehitaman melambangkan bahwa kedua *vertex* saling *visible*.

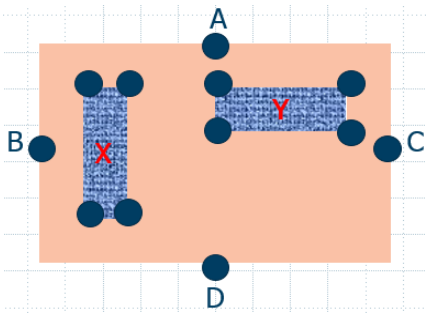
[1] http://www.spoj.com/problems/ARCHPLG

B. Prinsip Dasar Visibility Graph

Prinsip dasar dari *visibility graph* adalah sebagai berikut. Yang pertama, menganggap semua *interest point* menjadi sebuah *vertex*. Yang dimaksud dengan *interest point* adalah tiap titik pada pojok area halangan. Untuk lebih jelasnya, dapat dilihat pada Gambar 2 berikut.

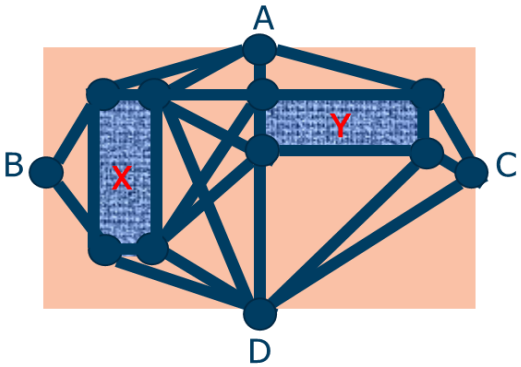


Gambar 1. Ilustrasi Visibilitas Antar Vertex



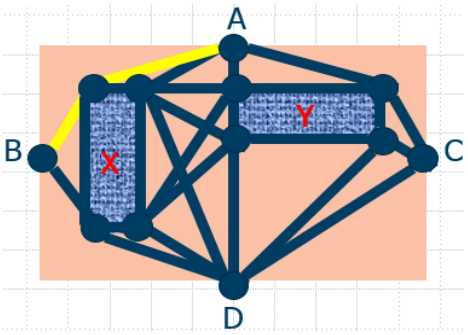
Gambar 2. Menganggap Semua Interest Point Menjadi Vertex

Kemudian, diperlukan pengecekan antar *vertex*, apakah dapat terbentuk suatu *edge* atau tidak. Apabila saling *visible*, maka berarti dapat terbentuk *edge* antar keduanya, begitupun sebaliknya apabila saling tidak *visible*, maka berarti tidak dapat terbentuk *edge* antar keduanya.



Gambar 3. Ilustrasi Hasil Pemanggilan Visibilitas Antar Vertex

Setelah dilakukan *visibility graph*, maka penentuan rute antar *vertex* akan lebih mudah, terutama untuk *vertex* yang awalnya saling tidak *visible*. Contohnya *vertex* A dan *vertex* B, meski tidak saling *visible*, namun dapat terhubung lewat beberapa *edge* / garis, salah-satunya dengan garis berwarna kuning pada Gambar 4 berikut.



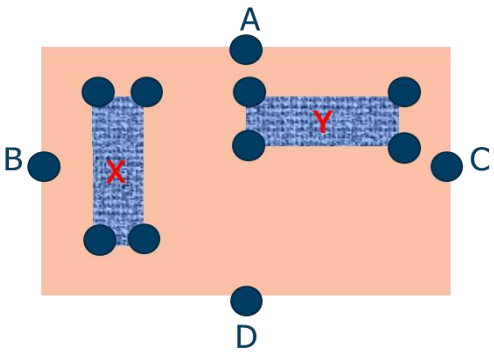
Gambar 4. Vertex A dan B dapat Terhubung

C. Visibility Graph Naive

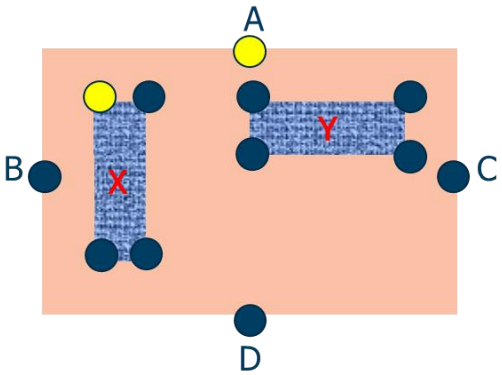
Visibility graph naive merupakan salah satu algoritma yang dapat menyelesaikan permasalahan *visibility graph*, dengan kompleksitas waktu sebesar $O(N^3)$. Algoritma ini dapat menentukan apakah suatu *vertex* dapat dijangkau dari *vertex* yang lain, dengan cara mencoba semua kemungkinan yang ada, yaitu mengecek tiap pasang *vertex* dengan tiap halangan yang ada. Berikut merupakan langkah-langkah dari *visibility graph naive*:

- Tiap *vertex* cek dengan *vertex* yang lain
- Bentuk garis lurus antara kedua *vertex* tersebut.
- Tiap garis tersebut cek dengan tiap halangan yang ada.
- Cek apakah garis lurus tadi bertabrakan dengan salah-satu halangan. Apabila:
  - Menabrak: diabaikan
  - Tidak menabrak : buat *edge* baru antara kedua *vertex* tersebut.

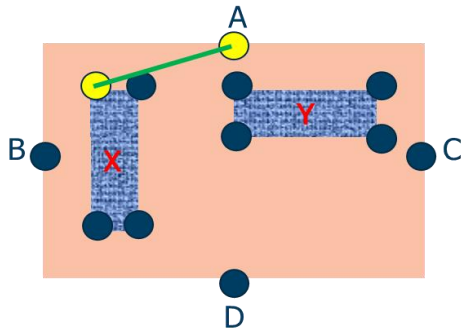
Ilustrasi iterasi algoritma tersebut dapat dilihat pada gambar-gambar berikut.



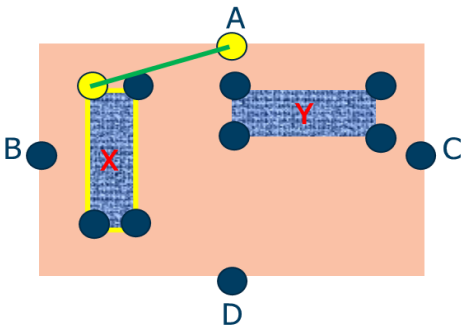
Gambar 5. Ilustrasi Kondisi Awal Graf



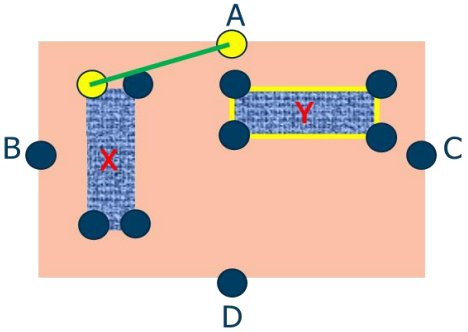
Gambar 6. Tiap Pasang Vertex dicek



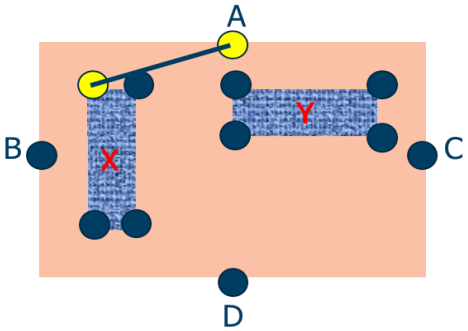
Gambar 7. Bentuk Garis Lurus Antara Kedua Vertex Tersebut



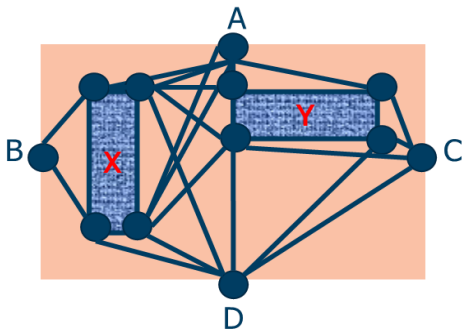
Gambar 8. Cek dengan Halangan 1



Gambar 9. Cek dengan Halangan **2**



Gambar 10. Bentuk Edge Antar Kedua Vertex



Gambar 11. Ilustrasi Hasil Pemanggilan Visibility Graph.

III. STRATEGI PENYELESAIAN

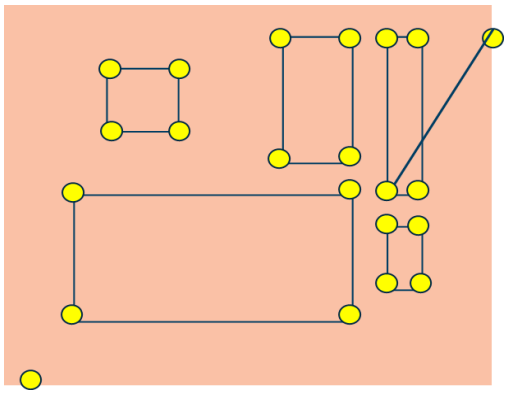
A. Penggunaan Visibility Graph Naive

Dikarenakan kompleksitas waktu yang cukup besar, maka diperlukan efisiensi kinerja dari algoritma ini. Beberapa efisiensi kinerja tersebut antara lain:

- Tidak semua *vertex* dicek visibilitasnya. Hanya *vertex* yang dikunjungi pada Dijkstra saja yang dicek visibilitasnya. Sehingga, algoritma ini hanya dipanggil saat pengecekan *current vertex* pada Dijkstra.
- Tiap *vertex* hanya akan dicek visibilitasnya terhadap *vertex* lain se-pulau.

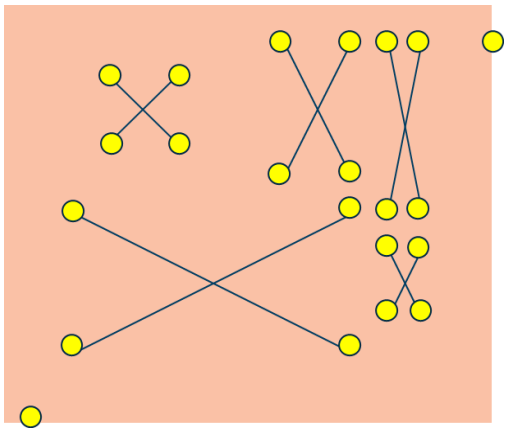
B. Penggambaran Entitas Halangan

Bentuk halangan pada permasalahan ini adalah persegi panjang. Untuk mempermudah pengecekan menabrak tidaknya, antara garis antar *vertex* dengan tiap halangan, maka halangan tersebut dapat digambarkan dengan 4 buah garis, dengan rincian 2 garis horizontal dan 2 garis vertikal. Sehingga garis antar *vertex* tersebut cukup dicek perpotongannya dengan 4 buah garis pada tiap halangan. Apabila terjadi perpotongan antara garis antar *vertex* dengan 1 atau lebih garis halangan, maka berarti tidak dapat terbentuk *edge* antara kedua *vertex* tersebut, begitupun sebaliknya.



Gambar 12. Pengecekan Garis Antar Vertex dengan Tiap Garis Halangan

Untuk mengurangi jumlah iterasi pengecekan tersebut, tiap halangan dapat digambarkan dengan 2 garis diagonal saja. Untuk ilustrasinya, dapat dilihat pada Gambar 13 berikut**.**



Gambar 13. Penggambaran Halangan dengan 2 Buah Garis Diagonal

C. Menghitung Total Jarak Rute dan Penyimpanan Posisi

Untuk menghitung tota jarak rute serta penyimpanan posisi yang dipilih pada rute terpendek, dapat digunakan nilai jarak pada tiap *vertex*, yaitu besar jarak dari tiap *vertex* terhadap *vertex* sumber, dan penyimpanan

informasi *vertex* asal (*previous*) dari tiap *vertex*. Setelah itu, dilakukan iterasi dari *vertex* tujuan hingga *vertex* sumber. Pada tiap iterasi:

- Jika *vertex* sekarang dan *vertex* asal:
  - berada pada satu pulau, maka selisih jaraknya ditambahkan ke total jarak pulau yang bersangkutan.
  - tidak berada pada satu pulau, maka selisih jaraknya ditambahkan ke total jarak kapal ferry.
- Nama *vertex* sekarang disimpan / dicetak.

IV. UJI COBA

A. *Uji Coba Kebenaran*

Uji coba kebenaran dilakukan dengan cara membandingkan hasil keluaran program terhadap permasalahan, serta dengan cara mengirimkan kode sumber program pada situs SPOJ dengan kode soal ARHPLG, nomor soal 780, dengan nama permasalahan "The Archipelago".

Gambar 14. Hasil Uji Coba pada Situs SPOJ

Kode sumber program mendapatkan umpan balik *Accepted* dengan waktu sebesar 0.26 detik, dan memori sebesar 10 MB. Selain itu, kode sumber mendapat peringkat pertama dari segi *running time*.

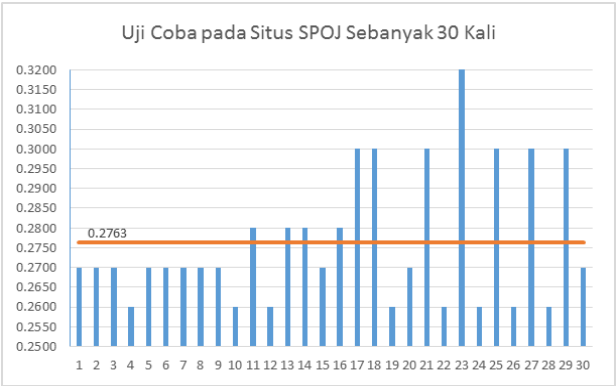
RANK	DATE	USER	RESULT	TIME	MEM	LANG
1	2016-04-10 09:54:17	Reva Yoga Pradana	accepted	0.26	11M	C++ 5

Gambar 15. Hasil Implementasi Mendapat Peringkat 1

B. *Uji Coba Efisiensi*

Uji coba efisiensi dilakukan dengan cara mengirim kode sumber pada situs SPOJ permasalahan "The Archipelago" sebanyak 30 kali. Hasil pengiriman kode sumber dapat dilihat pada Gambar 16.

Dari hasil tersebut, didapat bahwa waktu minimum berjalannya program adalah 0.26 detik, sedangkan waktu maksimumnya sebesar 0.32 detik. Rata-rata *running time* program adalah sebesar 0.2763 detik.



Gambar 16. Hasil Pengiriman Kode Sumber Sebanyak 30 Kali.

V. KESIMPULAN

Dari hasil penelitian ini, dapat diambil kesimpulan sebagai berikut:

1. Implementasi algoritma Dijkstra dan *visibility graph naive* dapat menyelesaikan permasalahan "The Archipelago" dengan benar.
2. Kompleksitas sebesar $O(|E| + VlogV)$ pada algoritma Dijkstra, dan kompleksitas sebesar $O(N^3)$ pada *visibility graph naive* cukup untuk menyelesaikan permasalahan "The Archipelago".
3. Optimasi-optimasi yang telah dilakukan dapat membantu mengurangi *running time* dari metode *visibility graph naive* yang sebenarnya memiliki kompeksitas cukup besar.
4. Waktu yang dibutuhkan program minimum 0,26 detik, maksimum 0,32 detik dan rata-rata 0,2763 detik.

UCAPAN TERIMA KASIH

Penulis bersyukur atas rahmat, ridha, dan hidayah Allah SWT kepada penulis selama hidup ini. Penulis juga berterimakasih yang sebesar-besarnya kepada orang tua, saudara, teman-teman, dosen pembimbing Pak Victor dan Pak Rully, serta dosen dan guru yang lain yang telah membimbing penulis selama ini. Penulis juga ingin berterimakasih kepada Muhammad Nuh, selaku Menteri dari Kemendikbud pada era Presiden Susilo Bambang Yudhoyono, yang telah memberikan program beasiswa Bidikmisi kepada penulis, sehingga penulis dapat menempuh kegiatan selama perkuliahan ini dengan sangat baik.

DAFTAR PUSTAKA

[1] O. C. M. v. K. M. O. Mark de Berg, Computational Geometry, 3rd penyunt., Berlin Heidelberg: Springer-Verlag, 2008.

[2] S. Halim dan F. Halim, Competitive Programming 2, Singapore: Lulu Publisher, 2011.

[3] Ĺ. Kuszner, "SPOJ.com - Problem ARCHPLG," 15 March 2006. [Online]. Available: http://www.spoj.com/problems/ARCHPLG/.

[4] Levitin, The Design & Analysis of Algorithms, 3rd penyunt., Pearson Education, Inc., 2012.